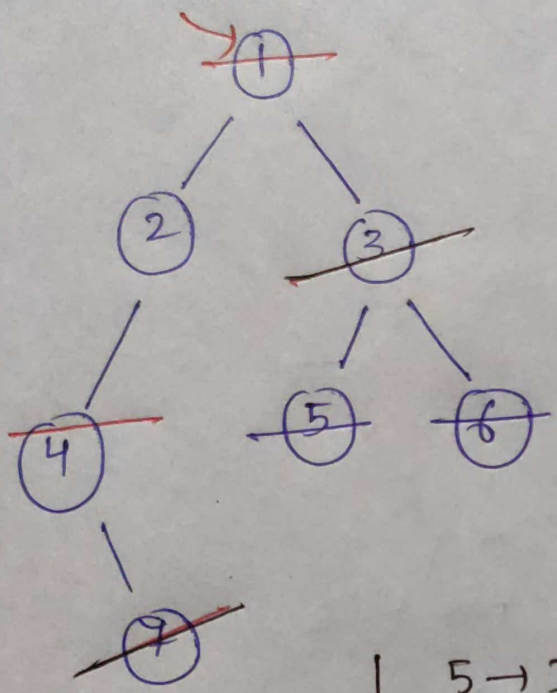
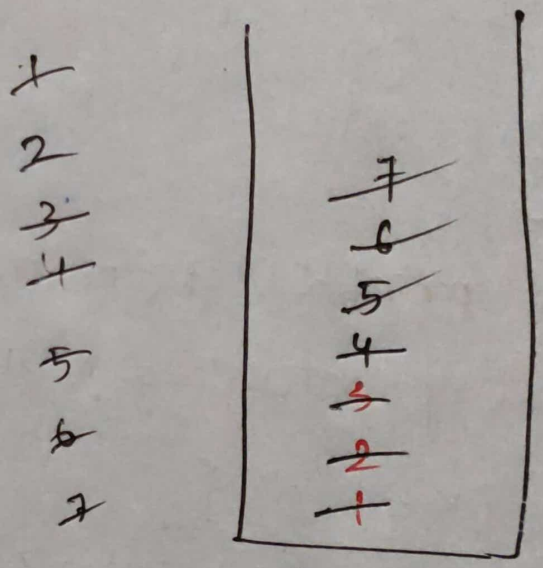
Min Time taken to Burn a Binary Tree from a node / Leaf Node.

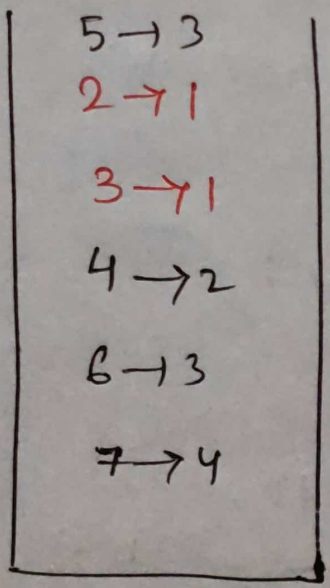


node = 2

3 seconds



Queue.



Travel complete.

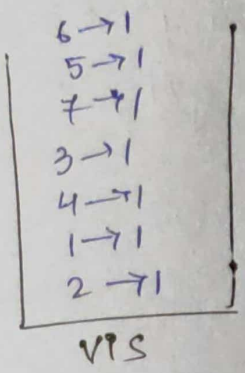
Check 1 → left ✓
→ right ✓

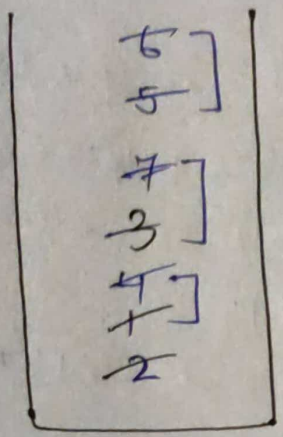
assign 2→1
parent

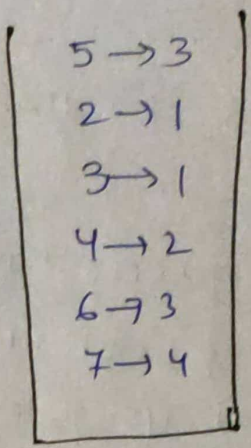
3→1

put left, right in queue

time = 0 1

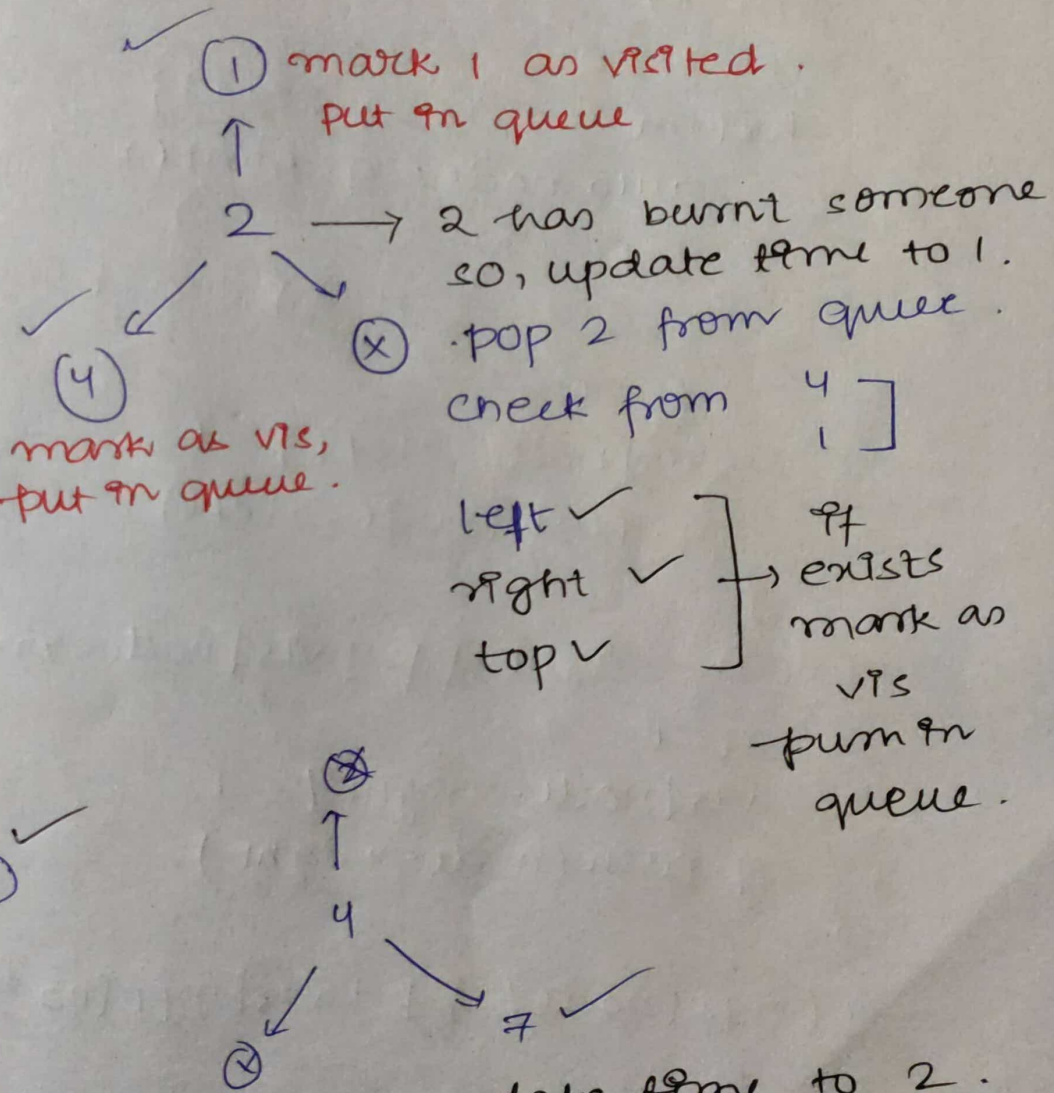






start from (node-2) mark as visited in vis ds.

check,



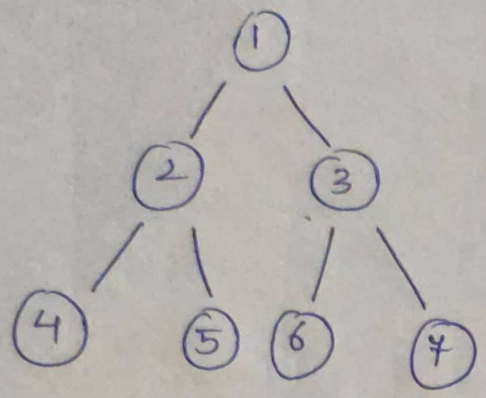
Similarly, we can check for further iterations.

Code:-

```
int findMaxDist(map<BinaryTree<int>*,BinaryTree<int>*>&mpp,
                BinaryTree<int>* target) {
    queue<BinaryTree<int>*> q;
    q.push(target);
    map<BinaryTree<int>*, int> vis;
    vis[target] = 1;
    int maxi = 0;
    while(!q.empty()) {
        int sz = q.size();
        int fl = 0;
        for(int i=0; i<sz; i++) {
            auto node = q.front();
            q.pop();
            if(node->left && !vis[node->left]) {
                fl = 1;
                vis[node->left] = 1;
                q.push(node->left);
            }
            if(node->right && !vis[node->right]) {
                fl = 1;
                vis[node->right] = 1;
                q.push(node->right);
            }
            if(mpp[node] && !vis[mpp[node]]) {
                fl = 1;
                vis[mpp[node]] = 1;
                q.push(mpp[node]);
            }
        }
        if(fl) maxi++;
    }
    return maxi;
}
```

```
BinaryTree<int>* bfsToMapParent(BinaryTree<int>* root,
map<BinaryTree<int>*, BinaryTree<int>*>& mpp,
int start) {
queue<BinaryTree<int>*> q;
q.push(root);
BinaryTree<int>* res;
while(!q.empty()) {
    BinaryTree<int>* node = q.front();
if(node->data == start)
        res = node;  q.pop();
  if(node->left) mpp[node->left]=node;
  q.push(node->left);
  if(node->right) mpp[node->right] = node;
  q.push(node->right);

  }
}
return res;
}

int timeToBurnTree(BinaryTree<int>* root,
int start) {
  map<BinaryTree<int>*, BinaryTree<int>*>
      mpp;
BinaryTree<int>* target = bfsToMapParents(root,
                                mpp, start);

int maxi = findMax(mpp, target);

return maxi;
}
```

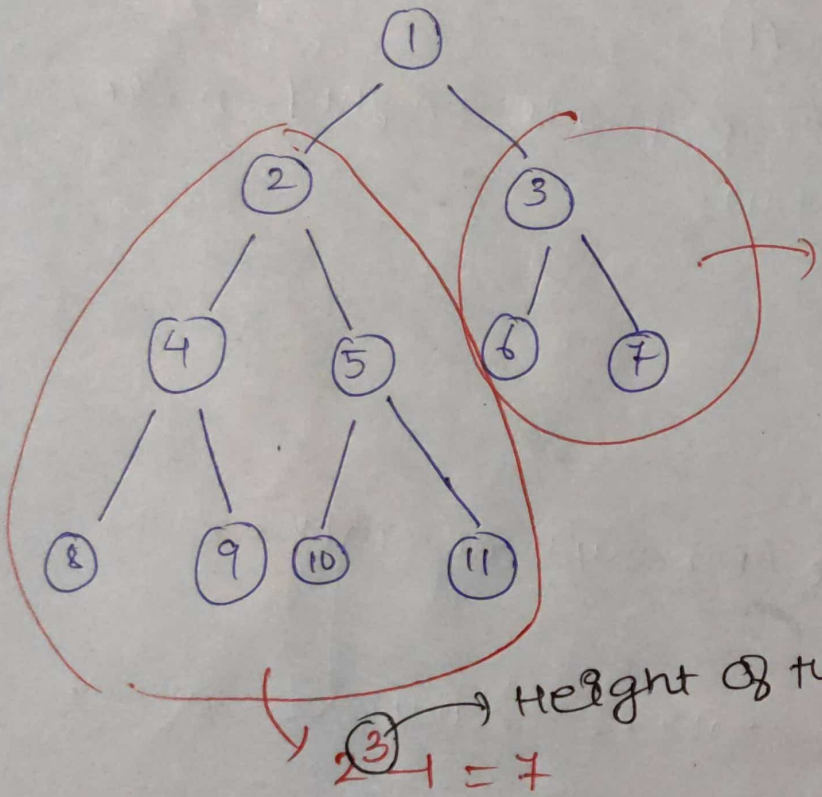
Count Complete Tree Nodes :-




```
morder (node, &cnt) {
    if (root == NULL)
        return;
    cnt++;
    morder (node -> left);
    morder (node -> right);
```

TC : O(n) (If all levels are filled)

SC : O(h)



1+7+3 = 11

$2^{2} - 1 = 3$

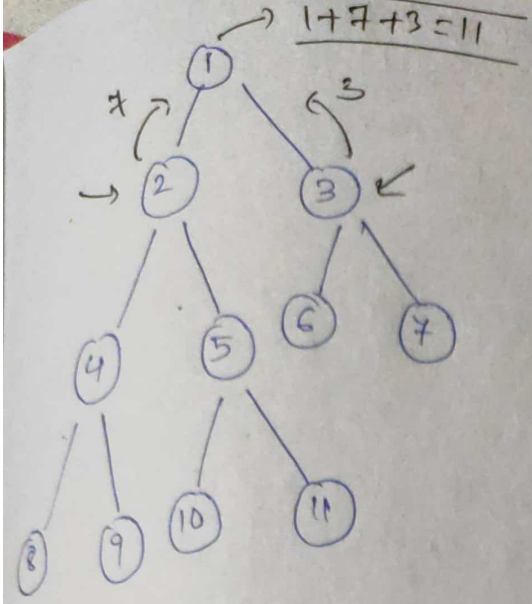
Height of the right subtree

Height of the left subtree

$2^{3} - 1 = 7$

1 + ( ) + ( )

lh rh



check from node-2

lh = 3
rh = 3
→ both are of same weight

that means its a complete binary tree.

$2^3 - 1 = 7$

lh = 2
rh = 2

$2^2 - 1 = 3$

Code :-

```
int CountNodes(TreeNode *root){
    if(root==NULL) return 0;

    int lh = findHeightLeft(root);
    int rh = findHeightRight(root);
    if(lh==rh) return (1<<lh)-1;   // 2^lh

    return 1+countNodes(root->left) +
           countNodes(root->right)
}

int findHeightLeft(TreeNode *node){
    int height = 0;
    while(node){
        height++;
        node = node->left;
    }
    return height;
}
```

↓ similarly for right

TC: $O((\log N)^2)$